CYPR-CD01169M US P

UNITED STATES PATENT APPLICATION FOR

AUTOMATIC GENERATION OF APPLICATION PROGRAM INTERFACES, SOURCE CODE, INTERRUPTS, AND DATASHEETS FOR MICROCONTROLLER PROGRAMMING

Inventors:

MANFRED BARTZ
MARAT ZHAKSILIKOV
STEVE ROE
KENNETH Y. OGAMI
MATTHEW A. PLEIS
DOUGLAS H. ANDERSON

Prepared by:

WAGNER, MURABITO & HAO LLP
TWO NORTH MARKET STREET
THIRD FLOOR
SAN JOSE, CALIFORNIA 95113
(408) 938-9060

AUTOMATIC GENERATION OF APPLICATION PROGRAM INTERFACES, SOURCE CODE, INTERRUPTS, AND DATASHEETS FOR MICROCONTROLLER PROGRAMMING

FIELD OF THE INVENTION

The present invention relates to the field of programmable systems on a chip (PSoCs). Specifically, the present invention relates to a software program which allows a circuit designer to configure a circuit and then automatically generate a series of items to facilitate programming a microcontroller.

BACKGROUND ART

Microcontrollers allow circuit designers great flexibility in design choice. However, programming the microcontroller to perform the desired functions can be an arduous task. Conventional software for programming microcontrollers is not very robust and does not offer designers many tools to reduce the amount of low level details they need to memorize in order to configure the chip.

Conventional software for programming microcontrollers is very difficult to use. In one system, many windows pop-up as the user attempts to program the microcontroller. Windows pop-up based on "flat-organized" drop down menus. Each window corresponds to a discrete function. However, many functions are required to do simple tasks. Consequently, the many displayed windows cause confusion because the user needs to keep track of which window is used for which function. Furthermore, it is very difficult to navigate between the windows because some windows overlap others. The user may

have difficulty remembering which windows contain what information and which windows receive what information.

Once a circuit designer selects the various functions desired for the circuit, the designer must organize those function within the constraints of the available resources of the hardware with which the design is to be implemented. Conventionally, the circuit designer manually places the functions within the available resources of a programmable device. Unfortunately, this process is tedious and error-prone.

The circuit designer must also design the various interconnections between the selected functions, as well as configure the input/output pins. Conventionally, this can be an arduous and error-prone process. For example, the circuit designer must map the functions he has selected to actual hardware. Multifunction input/output (I/O) ports or pins may be very difficult to configure. They typically have multiple registers that needed to be programmed to configure the pin type as well as the drive characteristics for each of the I/O pins.

Circuits designers also desire to have a datasheet describing the circuit he has designed. Conventionally, the datasheets are generated manually by the designers. Each time the design is modified, a new datasheet must be manually generated. Thus, the designer time is not used efficiently and the possibility of errors in the datasheet is great.

Finally, in many conventional systems, the microcontroller devices are programmed manually. The programmer needs to know all of the registers and other technical information required to instruct the microcontroller to do its embedded functions (e.g., start timing, stop timing, etc.). Manual programming is very error prone and tedious and difficult to error check.

Therefore, it would be advantageous to provide a method which provides for a convenient user-friendly interface for designing a circuit by programming a microcontroller. It would be further advantageous to provide a method which may help reduce errors in programming a microcontroller. Finally, it would be advantageous to provide such a method for programming a microcontroller which does not require the circuit designer to memorize registers and other technical information to invoke functions when programming a microcontroller.

Therefore, it would be advantageous to provide a method which may help reduce errors in programming a microcontroller. It would be further advantageous to provide such a method for programming a microcontroller which does not require the circuit designer to memorize register and other technical information to invoke functions when programming a microcontroller.

SUMMARY OF THE INVENTION

The present invention provides for a method to facilitate programming a microcontroller. Embodiments provide for a method which may help reduce errors in programming a microcontroller. Embodiments provide for such a method for programming a microcontroller which does not require the circuit designer to memorize registers and other technical information to invoke functions when programming the microcontroller. Embodiments provide an alternative to manually creating a datasheet describing a microcontroller implemented project. The present invention provides these advantages and others not specifically mentioned above but described in the sections to follow.

A method to facilitate programming a microcontroller is disclosed. In one embodiment, after a user configures the circuit by selecting circuit parameters and pin-outs, various items are automatically generated to facilitate programming the microcontroller. The generated items may include: application programming interfaces (APIs) for programming the operation of one or more user modules; source code for realizing the user modules in hardware; interrupt vectors to call interrupt service routines for one or more modules; and a data sheet for the circuit.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1A is a diagram illustrating a graphical user interface allowing a user to select a user module and display its schematic and its data sheet, according to an embodiment of the present invention.

Figure 1B is a diagram illustrating a graphical user interface allowing a user to place a user module in a graphical user interface, according to an embodiment of the present invention.

Figure 1C is a diagram illustrating a graphical user interface allowing a user to configure pins, according to an embodiment of the present invention.

Figure 1D is a diagram illustrating an editor workspace allowing a user to edit source code, according to an embodiment of the present invention.

Figure 2 is a flowchart illustrating steps of a process of facilitating programming a microcontroller, according to an embodiment of the present invention.

Figure 3A, Figure 3B, and Figure 3C are diagrams illustrating the position of a user module being iterated to new positions, according to an embodiment of the present invention.

Figure 4 is a diagram illustrating a graphical user interface allowing selection of user module parameters, according to an embodiment of the present invention.

Figure 5A, Figure 5B, and Figure 5C are diagrams illustrating graphical user interfaces for facilitating configuring I/O pins, according to an embodiment of the present invention.

Figure 6A, Figure 6B, Figure 6C, and Figure 6D are illustrations of graphical user interfaces for configuring interconnections between PSoC blocks, according to an embodiment of the present invention.

Figures 7A-7C are an exemplary file illustrating a source code table that may be automatically generated, according to an embodiment of the present invention.

Figures 8A-8B are an exemplary file showing source code that may be automatically generated, according to an embodiment of the present invention.

Figures 9A-9B are an exemplary file showing an application programming interface (API) that may be automatically generated, according to an embodiment of the present invention.

Figure 10 and Figure 11 are exemplary files showing application programming interfaces (APIs) that may be automatically generated, according to an embodiments of the present invention.

Figure 12A and Figure 12B illustrate an exemplary interrupt handler file which may be automatically generated, according to an embodiment of the present invention.

Figure 13A illustrates interrupt vectors that may be automatically generated in a startup file to build on interrupt vector table, according to an embodiment of the present invention.

Figure 13B is a table illustrating how the interrupt vectors of Figure 13A map to interrupts, according to an embodiment of the present invention.

Figure 14 is a schematic of a computer system, which may be used to implement embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of the present invention, a method for facilitating programming a microcontroller, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be recognized by one skilled in the art that the present invention may be practiced without these specific details or with equivalents thereof. In other instances, well known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

NOTATION AND NOMENCLATURE

Some portions of the detailed descriptions which follow are presented in terms of procedures, steps, logic blocks, processing, and other symbolic representations of operations on data bits that can be performed on computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, computer executed step, logic block, process, etc., is here, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "indexing" or "processing" or "computing" or "translating" or "calculating" or "determining" or "scrolling" or "displaying" or "recognizing" or "generating" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

To facilitate the design process, embodiments provide various work-spaces. For example, a user may move between a user module selection work-space, a user module placement workspace, and a user module pin-out work-space. Figure 1A illustrates an exemplary graphical user interface which allows a user to select user modules 304. Regarding user module selection, the workspace provides a user module window 302 for a catalog of available user modules 304, a listing 306 of selected user modules 304, a schematic 310 of a selected user module 304, plus its datasheet 308. The user may click on a user module 304 of user module window 302 to designate one. A histogram 350 (e.g., a series of cumulative bar charts or graphical indicators) of available resources on the target device (e.g., a microcontroller) is also shown.

The datasheet 308 is tabbed for easy navigation therethrough. The various windows may be displayed simultaneously for easy reference back-and-forth. Button 301 may be used to automatically generate source code for the project.

Referring now to Figure 1B, a user module placement work-space includes a resource graphic window 360 illustrating the placement of user modules 304 with respect to the available resources (e.g., available PSoC blocks 410 of a microcontroller) in a hardware layout graphical display. Throughout this application the term resource image may denote the blocks 410 upon which user modules 304 are placed in window 360. As the resource images may represent PSoC blocks in one embodiment, the resource images may be referred to as PSoC blocks for convenience. It will be understood that the resource images may represent other resources however, as the present invention is not limited to implementing the user modules 304 in PSoC blocks. Figure 1B shows a number of digital PSoC blocks 410a along the top row (e.g., the blocks labeled DBA00, DBA01, etc.), as well as four columns of analog PSoC blocks 410b (e.g., the blocks labeled ACA00, ACA01, etc.). The present invention is well suited to using any number of analog and digital PSoC blocks 410. Furthermore, the blocks in graphic window 360 are not limited to representing PSoC blocks.

A single user module 304 may map to one or more PSoC blocks 410. Color coding (not shown) may be used to relate the user modules 304 of selected modules window 306 with their schematic placement in resource graphic window 360. The analog 410b and digital 410a PSoC blocks may be

more generally defined as two different classes to which a user module 304 maps. The present invention is well-suited to having many different classes.

Referring now to Figure 1C, a pin-out configuration work-space is shown. The pin-out configuration work-space allows the user to connect PSoC blocks 410 to input/output (I/O) pins, as well as configure the I/O pins' drive characteristics. In one embodiment, a pin configuration window 380 may be used to configure pins. Pin configuration window 380 has a port column 381 a select column 382, and a drive column 383. In another embodiment, a user may to set pin configurations by clicking on the GUI of the chip 610. The operation of these features will be discussed more fully herein.

Referring now to Figure 1D, after the user has configured the device, embodiments automatically generate source code, which may be edited by the user. A directory window 366 (source tree window) provides a listing of various exemplary source code files and API files that may be automatically generated. An editor workspace 365 is provided for a user to edit various files. In this fashion, a user may program a microcontroller without having detailed knowledge of all the registers in the microcontroller.

Reference will now be made to the flowchart of Figure 2 and Figures 1A-1D and Figures 3-6D. Steps of process 200 may be implemented on a general purpose computer, such as illustrated in Figure 14. Referring now to step 210 of Figure 2 and Figure 1A, a selection 302 of available user modules 304 is displayed. A user module 304 may represent an accessible, pre-configured

function that once programmed and placed will work as a peripheral to a target device. For example, a user module may be an amplifier, pulse width modulator, counter, digital-to-analog converter, etc. The user module window 302 of Figure 1A shows amplifiers which may be selected. The user may select one of the 'buttons' 307 to cause the user module window 302 to display other user modules 304. (For example, to display Timers, Pulse Width Modulators, etc.).

Referring still to Figure 2 and Figure 1A, in step 220 in response to a user selection of one of the user modules 304, the selected user module 304 is displayed in a selected user module region 306 and a data sheet 308 and schematic 310 are displayed for the selected user module 306. Figure 1A shows a schematic 310 for an instrumentation amplifier, along with its datasheet 308. The user is allowed to add more user modules 304 to the design by selecting more user modules 304 and causing them to be added to the selected user module area 306.

Referring now to step 230 of Figure 2, in response to a request from a user for a potential (e.g., valid) position for a selected user module 304, a position is computed. The computer automatically determines the possible placements based on the available resources and the number of PSoC blocks 410 and the types of PSoC blocks 410 that are required for the unplaced user module 304. Because the user does not need to determine the potential placements, designing the circuit is faster and less error prone than conventional methods which do not provide such guidance.

User modules 304 may require multiple PSoC blocks 410 to be implemented. In some cases, user modules 304 may require special ports or hardware which may limit which PSoC blocks 410 can be used for their implementation. The process of mapping a user module 304 to PSoC blocks 410, such that the user module 304 is realized within the microcontroller, may be referred to as "user module placement." An embodiment automatically determines the possible placements of a user module 304 based on an Extensible Markup Language (XML) user module description and the hardware description of the underlying chip. However, the present invention is not limited to using XML descriptions. The potential placement positions may be automatically inferred based on the XML input data. Therefore, the placement process of embodiments of the present invention is data driven.

In step 240, one or more PSoC blocks 410 are highlighted to indicate a possible position for the user module 304 based on, for example, XML input data. The placement is shown in a graphical hardware layout diagram 360 by highlighting the PSoC blocks 410 involved. For example, referring to Figure 1B, the ADCINC12_1 user module 304 has been selected for placement in the window 360. This user module 304 requires two digital blocks 410a and one analog block 410b. The digital PSoC blocks 410a labeled DBA00 and DBA01 are highlighted to indicate a possible position for the ADCINC12_1 user module 304. Referring now to Figure 3A, the analog PSoC block 410a labeled ASB20 is highlighted to indicate that it is a possible position for the analog portion of the

ADCINC12_1 user module 304. Embodiments may use color coding to associate the highlighting color with a unique color assigned to that user module 304.

User module placement is described in co-pending US patent application serial number _____ , filed concurrently herewith, entitled "A SYSTEM AND METHOD FOR PERFORMING NEXT PLACEMENTS AND PRUNING OF DISALLOWED PLACEMENTS FOR PROGRAMMING AN INTEGRATED CIRCUIT," by Ogami et al., attorney docket number CYPR-CD01175M and assigned to the assignee of the present invention and incorporated herein by reference.

Referring now to Figures 3A-3C and to step 250 of Figure 2, after placing a user module 304, a user may desire to move it to another PSoC block 410 (or blocks). In step 250, a new possible position for a user module 304 is computed, in response to a user request for a new position for the user module 304. The user may select a next position button 371 to cause this to occur. Figures 3A-3C illustrate three possible positions for the analog portion of the ADCINC12_1 user module 304. The user may then click on a place module button 372 to place the module 304. Placements that are incompatible with the user module requirements (e.g., characteristics) are automatically pruned out by the software and therefore are not displayed as valid placements. In one embodiment, all positions are shown to the user, sequentially, each time the next placement icon 371 is selected. However, if a potential placement involves a PSoC block 410 that has already been used (e.g., by another placed user module 304), then in these cases the place user module 372 icon is grayed out indicating that this placement is only

valid if the resources were vacant. This allows the user to see all possible placements.

If a user module 304 consists of both and digital 410a and analog blocks 410b, the system may show next positions for the digital 410a and analog blocks 410b separately. Thus, the user may change the placement of one without affecting the other. For example, the position of the analog block 410b of the ADCINC12_1 user module 304 is moved in Figures 3A-3C. However, the digital blocks 410a for that module 304 do not move at this time. The user may separately seek a new position for those blocks 410 (e.g., digital blocks DBA00 and DBA01 in Figure 1B). Embodiments allow for multiple different classes to be separately placed. For example, rather than placing analog and digital blocks separately, the user may place memory, routing, and I/O separately in an embodiment which is not illustrated in the Figures. The present invention is well-suited to placing any number of classes separately. Furthermore, when placing a user module 304 with multiple classes, the system may highlight active resource images (e.g., those currently being placed) in a different color than inactive resource images.

The next position process is iterative, in that each time the next position button 371 is clicked, another possible placement will be determined and highlighted, etc., until the user module 304 is placed.

User module next placement is described in co-pending US patent application serial number _____ , filed concurrently herewith entitled

"SYSTEM AND METHOD FOR DECOUPLING AND ITERATING RESOURCES ASSOCIATED WITH A MODULE," by Ogami et al., attorney docket number CYPR-CD01180M and assigned to the assignee of the present invention and incorporated herein by reference.

Steps 210 through 250 may be repeated to allow the user to add more user modules 304. Each time a new user module is selected, a system resource window may be updated. Referring again to Figure 1A, for each user module 304 selected, the system updates the data in the Resource Manager window 350 with the number of occupied PSoC blocks 410, along with RAM and ROM usage used by the current set of "selected" user modules 304. The system may also prevent a user from selecting a user module 304 if it requires more resources than are currently available. Tracking the available space and memory of configurations for the design may be performed intermittently during the whole process of configuring the microcontroller. Embodiments provide a live graph tracking the PSoC blocks 410 used by percentage. The RAM and ROM monitors may track the amount RAM and ROM required to employ each selected user module 304.

After the user has selected one or more user modules 304, the user may select global parameters and user module parameters such as, for example, the gain of an amplifier, a clock speed, etc. Referring now to Figure 4 and to step 260 of Figure 2, in response to a user clicking on a region on a PSoC block 410 an interface 510 is displayed which allows the setting of user module parameters. For example, the user may place "the cursor" over the lower-left corner of a PSoC block 410 to set input parameters. The system may display a superficial chip or a

changed cursor in response to this. The user may then left-click a mouse, for example, to bring up a user module parameter window 510 to configure the user module input parameters. The process may be repeated in the lower-right corner of the PSoC block 410 for output parameters and on the upper-left corner for clock parameters. The present invention is not limited to these steps for bringing up a user module pop-up window 510, however. The system may then display the selected parameters in a user module parameter window 520. Various pop-up windows may be data driven in that the contents of the pop-up window may depend on, for example, the user module 304 selected. Alternatively, user parameters may be set in the user module parameter window 520.

When the user module 304 is placed (e.g., instantiated) on a particular PSoC block 410 the register settings and parameter settings may be mapped to a physical register address on the chip. This may also associate interrupt vectors that the user module 304 uses based on the PSoC block 410. Each of the digital blocks 410a maps to one vector and each column of analog blocks 410b maps to one vector. Once the user modules 304 are placed and the parameters are set, all the physical address registers that are associated with that user module 304 are fixed and the register values are determined.

In addition to setting user module parameters, the user also may set global parameters. For example, referring still to Figure 4, a global resource window 370 is seen. Global resources may be hardware settings that determine the underlying operation of the part (e.g., the CPU_Clock, which may designate the speed in which the microcontroller processes). These settings may be used to program

global registers, which may be in addition to the registers set by configuring the user module parameters.

One embodiment provides for a graphical user interface for facilitating the configuration of I/O pins in a microcontroller software design tool. Referring now to Figures 5A-5C and to step 270 of Figure 2, a GUI is displayed to allow input/output pins to be configured. Each pin has a pin number associated therewith. Referring to Figure 5A, when the user clicks near a pin of GUI 610, a small window 375 opens allowing the pin type (e.g., Port_0_1) and drive type (e.g., Port_0_1_Drive) to be configured. Referring now to window 620 of Figure 5B, the pin type may include analog input or analog output or global bus, etc. Referring now to window 630 of Figure 5C, the drive type may include high-z, pull-up, pull-down, strong, etc. The windows 620 and 630 may include a list that contains items that can be selected using the cursor. When the cursor is clicked outside of the windows 620 or 630, then the windows 620, 630 disappear automatically.

In another embodiment, a pin parameter table is provided to configure the pins. Referring to Figure 5A, the pin parameter table 380 includes a column for pin number 381, pin type 382 and drive type 383. The entries in the pin parameter table 380 can be selected by the cursor and again, the relevant window will open so that pin type or drive type can be selected. Therefore, the GUI of the chip 610 or the pin parameter table 380 can be used to configure the pins.

Each pin may contain three register values for configuration of both pin type and drive type. By using this user interface, the user need not be concerned with remembering register values, etc., for configuring the pins. Further, the user need not worry about how the configuration is to be done using the registers.

Pin configuration is described in co-pending U.S. patent application serial number _____ , filed October 29, 2001, entitled "PIN-OUT CONNECTIONS/DRIVE LEVELS DIRECT-SET BY DROP DOWN LIST," by Ogami et al., attorney docket number CYPR-CD01173M and assigned to the assignee of the present invention and incorporated herein by reference.

Referring now to Figure 6A-6D and to step 280 of Figure 2, embodiments provide many different windows to assist the user in setting various parameters to specify interconnectivity of PSoC blocks 410. Referring to Figure 6A, the user may cause window 605 to appear to configure the analog output buffer. Referring to Figure 6B, the user may cause a clock window 606 to appear by clicking on a clock MUX 616 to configure which clock will be the input to a column of analog PSoC blocks 410b. Referring to Figure 6C a port selection window 607 is shown. The port selection window 607 may be made to appear by clicking on or near the pin input MUX 608. The user may then select the input port. Referring now to Figure 6D, the user may click on or near the analog clocking MUX 614 to cause a window 613 to appear to select which digital PSoC block 410a should be selected by the clock MUX (616 of Figure 6B).

Referring now to step 290 of Figure 2, after the circuit has been configured by the user, the system automatically generates Application Program Interfaces (APIs), source code to implement the user's design, a data sheet of the user's design, and interrupt vectors. For example, referring to Figure 1A, the user clicks on the generate application code button 301. The system may use all device configurations to update existing assembly-source and C compiler code and generate Application Program Interfaces (APIs) and Interrupt Service Routine (ISR) shells. At this time, the system also creates a data sheet based on the part configurations. Embodiments produce files that are suitable for use with emulators and debuggers to allow these configurations to be emulated and debugged in a simple and convenient fashion. This concludes the discussion of the steps of Figure 2.

Source Code Files

Embodiments automatically generate source code files for realizing the user modules within the PSoC blocks 410 in the chip. Throughout this application the term source code may be defined as the code that is used to program registers on the chip to implement the selected user modules 304 as they have been placed by the user and to configure the PSoC blocks to operate with the user selected parameters, interconnections, and pin-outs. Thus, automatically generated source code programs may realize the user modules 304 within the PSoC blocks 410.

The automatically generated files may be programmed into flash memory of a target device (e.g., a microcontroller). The flash memory may then be used to program registers on the target device in order to implement a particular user module 304 in hardware (e.g., actual PSoC blocks 410). Therefore, the source code may be generated based on the selection, placement, and configuration of the user modules 304. For example, the automatic code generation processes take into account the parameterization of the user modules 304 and the placement of the user modules 304, which the user may perform via a GUI of embodiments of the present invention.

By using the automatically generated source code, a user need not be aware of all of the low level registers and other low level technical information required to program a microcontroller to realize the user modules 304. The user need only to interact with the GUIs which are written in a higher level of abstraction.

Exemplary source code files are included in Figures 7A-7C and Figures 8A-8B. The exemplary file of Figures 7A-7C is a data table which is used to set register values on the target device based on the user's configuration. The exemplary file of Figures 8A-8B moves the data from the file of Figures 7A-7C into memory (e.g., flash) on the target device.

Automatic generation of source code files is described in co-pending US patent application serial number _____ , filed November 15, 2001, entitled "DESIGN SYSTEM PROVIDING AUTOMATIC SOURCE CODE

GENERATION FOR PERSONALIZATION AND PARAMETERIZATION OF USER MODULES," by Ogami, attorney docket number CYPR-CD01177M and assigned to the assignee of the present invention and incorporated herein by reference.

APIs

After the microcontroller has been configured to implement the selected user modules 304, the user may wish to program desired functionality into the microcontroller. For example, a user may wish the user modules 304, now implemented in PSoC blocks in hardware, to function in a certain fashion when the microcontroller is being used. Embodiments automatically generate APIs, which can be used to perform common functions that are required to interact with the user module (e.g., how to start the timer, how to stop the timer, how to talk to the timer, etc.) from an application program level. For example, a user may insert an API into a software program which he writes. Thus, one type of API may be described as a function call; however, APIs are not limited to function calls.

By using automatically generated APIs, a user need not be aware of all of the low level registers and other low level technical information required to instruct the user module to invoke its functions. The user need only to interact with the APIs which are written in a higher level of abstraction.

In one embodiment, the API files that are automatically generated include the name of the instantiation of the user module 304 for which they are

associated. This makes it easy for the user to keep track of the files. The application editor workspace 365 allows easy viewing, presentation, and editing of these files. A directory window 366 (source tree window) provides a hierarchical ordering of these and other files by file type and user module.

Automatic generation of API files may take into consideration the configuration the user made. For example, the values to which certain registers are set may depend on which block 410 the user module 304 is placed in. Figure 9A and Figure 9B illustrate an exemplary API file that is automatically generated for a user module ADCINC12_1. This file contains the actual code for the API. Figure 10 and Figure 11 illustrate two more exemplary API files that may be generated for a user module named ADCINC12_1. Each instantiation of each user module 304 may have such files automatically generated for it. The file in Figure 10 may be used to allow programs written in the C programming language to access the user module APIs. The file of Figure 11 contains equates that are used by the APIs. The APIs simplify the designer's task by declaring values for various registers based on the PSoC blocks 410 the user module 304 occupies.

Additionally, the API files may be conditionally compiled based on the parameter selections the user made during device configuration. For example, a user module 304 for a digital-to-analog converter may have a parameter for the data format that allows three choices such as: "offset_binary", "twos_complement", and "two_byte_sign_and_magnitude". An API file may

be generated by conditionally compiling the code based on the parameter selected.

Automatic generation of APIs is described in co-pending US patent application serial number _____ , filed concurrently herewith, entitled "AUTOMATIC API GENERATION TO FUNCTIONAL PSOC BLOCKS," by Ogami et al., attorney docket number CYPR-CD01198M and assigned to the assignee of the present invention and incorporated herein by reference.

Interrupt Service Routines

ISRs (Interrupt Service Routines) may also be automatically generated during the device configuration. These may be shells or routines to provide the device-interface and interrupt-activity framework for source programming. Thus, interrupt service routines may be described as a type of API. Automatic generation of ISRs may be performed each time device application code is generated and is transparent to the user.

Figure 12A and Figure 12B illustrate an exemplary interrupt handler file which is automatically generated for a user module 304 for an analog-to-digital converter whose name is ADCINC12_1. The exemplary ISR contains some code because it performs part of the user module function. Other ISRs may be shells with no code automatically generated. The user may add code to either the shell ISR or the partially filled ISRs.

Interrupt Vectors

Embodiments automatically generate interrupt vectors, based on the placement of the user modules 304 on the PSoC blocks 410 during device configuration. In one embodiment, there are two types of interrupt vectors: fixed function and configurable PSoC blocks 410. Examples of fixed function interrupts may be: Reset, Supply Monitor, and Sleep Timer. The present invention is well-suited to a variety of other fixed function interrupts.

In the present embodiment, the configurable PSoC block interrupts may include eight digital block interrupts and four analog column block interrupts. The present invention is well-suited to other numbers of configurable PSoC block interrupts. The definition of a configurable PSoC block interrupt (e.g., the ISR that is called) depends on the user module 304 that occupies that block 410. The present invention is not limited to automatically generating interrupt vectors for PSoC blocks.

Thus, embodiments may build an interrupt vector table. In one embodiment, a call or jump to the user module's interrupt handler is inserted in a startup source file. The startup file may be run when the target device is re-booted. In this fashion, the proper interrupt handler will be called when its interrupt occurs during target device operation. Figure 13A illustrates an interrupt vector table 1300 with code which has been automatically added. In particular, the "ljmp" instructions 1305 have been added.

The exemplary table 1350 in Figure 13B shows how the startup source file vector names 1351 map to the data sheet interrupt names 1352 and to fixed and PSoC block (configurable) interrupts 1353. For example, interrupt 05 maps to digital PSoC block 410a "DBA03". Furthermore, it is of type PSoC block. Referring to Figure 13A, a ljmp to the counter_16 ISR has been automatically placed in the vector table 1300 at the location for PSoC block DBA03. Referring now to Figure 1B, the ljmp was placed automatically to reflect the configuration having module 304 CNTR16_MSB at digital block 410b "DBA03". Other interrupt vectors have been automatically added to the interrupt vector table 1300, as well.

Data Sheets

Embodiments may automatically generate a datasheet. The user module 304 and device descriptions may be stored in XML format. Parameterization information regarding the project may be combined with the XML user module and XML device descriptions to generate an HTML output datasheet for the design. Any output format can be used. An Extensible Stylesheet Language (XSL) extension may be used to perform the combination according to a predetermined stylesheet that represents the datasheet. However, the present invention is not limited to XML, XSL, and HTML to implement automatically generated datasheets.

The datasheet in HTML can be viewed by a browser and may include expected information of a datasheet including pin-out information, schematics, connectivity, parameters, block information, signal information, etc.

By using XSL to generate the HTML output (datasheet) based on XML descriptions, embodiments are very adaptable to changes in any of the user module 304 descriptions. For instance, the automatic datasheet generation process can readily be adapted to new user modules 304 by the mere addition of their XML files into a user module library. No recompiling of the software tool is required.

Automatic generation of datasheets is described in co-pending US patent application serial number _____ , filed concurrently herewith, entitled "SYSTEM AND METHOD FOR DYNAMICALLY GENERATING A CONFIGURATION DATASHEET," by Ogami et al., attorney docket number CYPR-CD01174M and assigned to the assignee of the present invention and incorporated herein by reference.

Figure 14 illustrates circuitry of computer system 100, which may form a platform for embodiments of facilitating programming of a microcontroller. Computer system 100 includes an address/data bus 99 for communicating information, a central processor 101 coupled with the bus for processing information and instructions, a volatile memory 102 (e.g., random access memory RAM) coupled with the bus 99 for storing information and instructions for the central processor 101 and a non-volatile memory 103 (e.g., read only memory ROM) coupled with the bus 99 for storing static information and instructions for the processor 101. Computer system 100 also includes an optional data storage

device 104 (e.g., a magnetic or optical disk and disk drive) coupled with the bus 99 for storing information and instructions.

With reference still to Figure 14, system 100 of the present invention also includes an optional alphanumeric input device 106 including alphanumeric and function keys is coupled to bus 99 for communicating information and command selections to central processor unit 101. System 100 also optionally includes a cursor control device 107 coupled to bus 99 for communicating user input information and command selections to central processor unit 101. System 100 of the present embodiment also includes an optional display device 105 coupled to bus 99 for displaying information. A signal input/output communication device 108 coupled to bus 99 provides communication with external devices.

The preferred embodiment of the present invention, a method for automatically generating APIs, Interrupt Vectors, source code for programming a microcontroller, and a datasheet for the circuit implemented in the microcontroller, is thus described. While the present invention has been described in particular embodiments, it should be appreciated that the present invention should not be construed as limited by such embodiments, but rather construed according to the below claims.